Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer

Catherine Meadows
Code 5543
Naval Research Laboratory
Washington, DC 20375
meadows@itd.nrl.navy.mil

Abstract

In this paper we show how the NRL Protocol Analyzer, a special-purpose formal methods tool designed for the verification of cryptographic protocols, was used in the analysis of the Internet Key Exchange (IKE) protocol. We describe some of the challenges we faced in analyzing IKE, which specifies a set of closely related subprotocols, and we show how this led to a number of improvements to the Analyzer. We also describe the results of our analysis, which uncovered several ambiguities and omissions in the specification which would have made possible attacks on some implementations that conformed to the letter, if not necessarily the intentions, of the specifications.

1 Introduction

The Internet Key Exchange protocol (IKE) is a key exchange protocol being developed by the IP Security Protocol (IPSEC) Working Group of the Internet Engineering Task Force (IETF). It is intended to provide the security support for client protocols of the Internet Protocol. As such, it does much more than simply distribute keys; it also is intended to be used to establish *Security Associations* that specify such things as the protocol format used, the cryptographic and hashing algorithms used, and other necessary features for secure communication. Since it is intended to be flexible, it supports a number of different types of key exchange options, including digital signatures, public key encryption, and conventional encryption using shared keys. The Diffie-Hellman algorithm is used to generate shared key material, but is optional in some cases.

IKE has evolved from a number of different protocols, including ISAKMP [14], Oakley [22], the Station-to-Station protocol [4], and SKEME [10], the last two of which influenced the development of Oakley. Although the ISAKMP protocol has been analyzed by Millen [20] using formal methods, IKE itself had not previously been subjected to formal analysis[1]. Yet such analysis is important, not only because IKE introduces new features that go beyond the original protocols, but because the possible interaction of the different subprotocols, which use similar formats, could lead to new insecurities. Indeed, this has been shown to be the case for other protocols of this type. For example, Benaloh at al. [2] point out an attack on an early version of SSL in which an intruder can pass off a compromised weak key as a strong key by interleaving two subprotocols, one of which uses the weak key, and one of which is intended to use a strong key. Kelsey and Schneier [8] show that, given an initial protocol, it is always possible to construct one with which it will interact insecurely. Thus having an analysis of IKE itself, as well as of its ancestors, is important.

Having a formal analysis of IKE is not only useful in providing a greater understanding of the protocol itself, but also because it can help extend the frontiers of the application of formal methods to protocol analysis. Most applications of formal methods to cryptographic protocols have concentrated on the analysis of single message sequences. Concurrent, sequential, and interleaved communications involving either the same or other parties are also considered, but in each case the set of actions taken is the same for all parties playing the same role. Most real-life protocols, however, offer a number of different optional subpro-

[1] We should note, however, that a belief logic analysis of IKE appeared as this paper was being written [13].

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1999		**00-00-1999 to 00-00-1999**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Code 5543,4555 Overlook Avenue, SW,Washington,DC,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**16**	

tocols which could possibly interact with each other in harmful ways. Extending existing formal methods techniques to handle such potential interactions risks the possibility of intractable state explosions. This is probably the reason why until now little work has been done on this problem (other than to define sub-cases for which it is tractable [6, 5]), with a few notable exceptions, such as Paulson in [24], who, although he does not explicitly address the sub-protocol issue, does consider the case of a single protocol with optional messages and fields.

A formal analysis of IKE is of interest for yet another reason. Although IKE had not previously been the subject of any formal analysis, it has undergone the usual vetting procedure for Internet standards: that is, it has been published on the World Wide Web and subjected to intense scrutiny by experts. Thus it has received much more informal analysis than most cryptographic protocols. A formal analysis of IKE gives us the opportunity to compare the results of formal with informal analysis: are there types of problems that the formal analysis would catch that the informal analysis would not, and vice versa?

In this paper we describe how the NRL Protocol Analyzer, a tool based on a combination of state exploration and theorem proving techniques, was used to the analyze IKE. We show the modifications that needed to be made to the Analyzer before it could be applied to protocols like IKE, and we describe the results we obtained. We also describe some changes to the IKE specification that we suggested as a result of our analysis, and consider possible further improvements that could be made to the NRL Protocol Analyzer.

Our formal analysis caught several problems with the IKE specification. These problems were not so much flaws in the protocol design itself, but rather ambiguities and omissions in the specification that could lead to insecure or incorrect implementations if the specification were not correctly understood. Indeed, in one case we found that several incorrect implementations were being developed as a result of an omission in the IKE specification.

The rest of the paper is organized as follows. In Section 2 we describe the IKE protocol. In Section 3 we give a brief overview of the NRL Protocol Analyzer. In Section 4 we describe the modifications that needed to be made to the Analyzer before it could be successfully applied to protocols like IKE. In Section 5 we describe our specification of IKE. In Section 6 we describe the analysis and its results. Section 7 concludes the paper and discusses some issues that were raised by this work.

2 The IKE Protocol

The IKE protocol can be thought of as a combination of two protocols, the Internet Security Association and Key Management Protocol (ISAKMP) [14] and Oakley [22]. ISAKMP provides a framework for establishing security associations and cryptographic keys, but does not prescribe any particular authentication mechanism. Indeed, ISAKMP is supposed to be integrable with a number of security protocols. Oakley, on the other hand, is a suite of key agreement protocols in which two parties generate a key jointly. The IKE document describes how Oakley can be used to provide an instantiation of ISAKMP.

A typical key establishment protocol proceeds in one phase, in which two parties use master keys to establish shared keying material. IKE, however, proceeds in two such phases. In the first phase, two entities use master keys to agree, not only on keying material, but on the various mechanisms (e.g. cryptographic algorithms, hash functions, etc.), that they will use in the second phase. The keying material and set of mechanisms thus agreed upon is called a *security association*. In the second phase, the keys and mechanisms produced in the first phase are used to agree upon new keys and mechanisms (that is, new security associations) that will be used to protect and authenticate further communications. The security association established in Phase One is bidirectional, so the initiator in the first phase can be either initiator or responder in the second phase.

Oakley may be used in one of four different modes: main mode, aggressive mode, quick mode, and new group mode. These offer different types of services. Both main and aggressive mode can be used in Phase One. In main mode, certain types of identifying information will not be exchanged until some initial authentication has occurred. In aggressive mode, this level of protection is not provided, but the exchange is accomplished in fewer messages. Main and aggressive mode can be implemented in four different ways: using shared keys, signatures, or public keys in two different ways for authentication. In all of these, the Diffie-Hellman protocol is used to generate the keying material.

Quick mode is used as part of the Phase Two negotiation process. In quick mode, the key material generated in Phase One is used to encrypt and authenticate messages used to generate the key material produced for Phase Two. Quick mode offers Diffie-

Hellman key exchange as an option that can be used to provide perfect forward secrecy; the other option is to use more conventional shared key generation mechanisms, which, while they are faster, do not provide the same degree of security. In IKE, identity information is only required in Quick Mode messages when ISAKMP is acting as a client negotiator on behalf of another party (for example, hosts negotiating security associations for applications or users) ; otherwise identities may be assumed to be the IP addresses of the ISAKMP peers. Finally, new group mode is used to agree on a new Diffie-Hellman group.

As we can see, IKE is really a combination of a number of different subprotocols. Phase One offers a choice of eight different subprotocols, with two choices for mode, and four choices for authentication mechanism. Phase Two offers a choice of four different subprotocols, depending upon whether perfect forward secrecy is or is not provided, and depending upon whether or not explicit identity information is included. Thus, all in all, when the new group protocol is included, thirteen different subprotocols are offered.

3 The NRL Protocol Analyzer

The NRL Protocol Analyzer is a formal methods tool for analyzing security properties of cryptographic protocols. It is unique among tools that have been applied to the analysis of cryptographic protocols in that it uses automatic invariant generation to limit a potentially infinite search space in combination with exploration of the remaining space to generate attacks on insecure protocols and provide security proofs for secure ones, even in the face of a potentially unlimited number of protocol executions or an unlimited number of intruder actions. In this combination of methods it probably most resembles the Stanford Temporal Prover (STeP) [12], although it and STeP are applicable to somewhat different problem domains.

Protocols in the Analyzer model are specified as communicating state machines, one of which is a hostile intruder who can read all traffic, modify or delete traffic, perform cryptographic operations, and may be in cooperation with some legitimate users of the system.

Each honest participant is represented as a single state machine, each of which possesses a set of local state variables. These local state variables are particular to the particular protocol specification.

A local execution is defined with respect to a party, plus a role (e.g., initiator or responder), plus the particular local execution of the protocol. We refer to a local execution as a *round*. Each round is identified by a variable referred to as a *round number*.

The user of the Analyzer attempts to determine whether or not a protocol is secure by specifying an insecure state. The Analyzer works backwards from that state until it it has explored the search space exhaustively, so that each path produced either begins in an initial state (describing an attack) or an unreachable state.

The Analyzer determines whether a protocol rule could be used to produce a state by means of a process known as *narrowing*. Terms (that is expressions made out of variables, constants, and function symbols) used in protocol specifications are assumed to obey a set of explicitly defined rewrite rules, that is, equations such that the right-hand side of the equation is "simpler" than the left-hand side according to some well-defined measure. An example of a rewrite rule would be one saying that the result of encrypting a term and then decrypting it with the same key reduces to the original term. Terms used in state descriptions are assumed to be irreducible (no further rewrite rules apply), while terms used as output of rules may possibly not be reducible. The narrowing algorithm is used to find all substitutions to the variables involved such that the terms in the rule output become reducible to the terms in the state description. The narrowing algorithm is very dependent upon the fact that the identities used are rewrite rules, as well as the fact that they obey certain other well-defined properties.

The Analyzer makes no assumptions about limits on the number of protocol executions, the number of principals performing the different executions, the number of interleaved executions, or the number of times cryptographic functions are applied. This results in a search space that is originally infinite. However, the Analyzer provides means for specifying and proving inductive lemmas about the unreachability of infinite classes of states. This allows the user to narrow down the search space so that in many cases an exhaustive search is possible. These inductive lemmas are formulated in terms of formal languages. The user gives the Analyzer a seed term, and the Analyzer uses the seed term to construct a language and prove that, if the intruder learns a term in the language, then it must have already known a term in that language, thus inductively proving that the intruder can never learn a term in the language. For protocols involving public and shared-key encryption, we have developed a standard set of seed terms : master keys, encrypted data where the data is not known by the intruder, decrypted data where the term decrypted is not known

by the intruder, concatenation of two terms where one of the terms is not yet known by the intruder, and signed data.

The Analyzer also possesses another means of limiting the search space. If the Analyzer proves that a certain state is unreachable, or reachable under certain conditions, it can store that fact in a database using a tool called the State Unifier. From then on, whenever the Analyzer encounters that state during a search, it will discard it as unreachable if the state was proved unreachable, or attempt to prove that the conditions can hold if the state was proved reachable only under certain conditions. Until recently, much of the work involved in using the Analyzer involved identifying which states found by it would provide most useful input to the State Unifier.

The Analyzer has been applied to a number of different cryptographic protocols, and has found flaws in several. In some cases the flaws had not been discovered before. Examples of protocols the Analyzer has been used to examine are the Simmons Selective Broadcast Protocol [16], the Burns-Mitchell Ticket Granting Protocol [15], and an early version of the Encapsulating Security Protocol [26]. A more detailed description of the Analyzer is given in [17].

4 Improvements Made to the Protocol Analyzer

In order to analyze a protocol the size of IKE, it was necessary to make a number of improvements to the Protocol Analyzer. We should point out, however, that although these improvements made the analysis of IKE possible, they were not designed with IKE alone in mind, but rather any complex protocol that offers multiple options. In particular, we have also been performing an analysis of the Secure Electronic Transactions Protocol (SET) [1], a protocol even more complex than IKE, and we have found these improvements equally helpful there.

Improvements that were made to the Protocol Analyzer can be divided into three parts. These are: improvements to the Protocol Analyzer language, improvements to the theorem prover, and improvements to the query structure. We will describe each of these in more detail below.

4.1 Improvements to the Specification Language

The most obviously necessary changes were to the Protocol Analyzer specification language. The original language specified a set of possible state transitions. Conditions for a transition to fire were set on messages received and on values of state variables. Outputs of transition were messages sent and new values of state variables. Although the language was expressive enough to specify a protocol like IKE, it was low-level enough so that it was difficult to write and understand a protocol that involved many choices and/or a long sequences of state transitions. Thus the language was modified to include more high-level constructs such as if-then-else and subroutines. The representation of state variables and messages was also cleaned up and simplified.

This made it possible to specify the execution of different subroutines in the following way. An initial transition would be specified in which the initiator would nondeterministically choose a subprotocol. Then, depending upon the subprotocol it chose, it would execute the subroutine appropriate to that protocol. Subroutines would also be defined for each subprotocol for the responder. The responder would then chose which subprotocol to execute depending on the message it received from the initiator.

For example, the initiator's choice of whether or not to use aggressive or main mode uses the following transition:

```
Subroutine
   init_choosemode(user(A,honest),N,T):
init_kea := mainmode,
Or:
init_kea := aggressive.
```

The initiator later on uses the value of the init_kea state variable to decide whether to execute the init_keymain subprotocol (corresponding to main mode), or the init_sendsaggsa subprotocol (corresponding to aggressive mode):

```
Subroutine init_sendsa(user(A,honest),N,T):
If:
verify({init_kea},mainmode),
Then:
send msg(user(A,honest),{init_dest},
  [{init_cookie},nil,
     sa,{init_kea},nil,nil,{init_situation},
     {init_doi},
     nil,
     proposal1,
     nil,
     init_authtype},{init_plist}],N),
init_sendskeymain,
Else If:
verify({init_kea},aggressive),
```

```
Then:
init_grp := choice(one,grp,{init_plist}),
init_sendsaggsa.
```

When the Analyzer is searching backwards, and it comes to a state in which an input message is specified, it will examine all transitions that produce output messages. Any transition producing an output message that satisfies the requirements of the input message can be used to produce a previous state, no matter which subprotocol it corresponds to. This allows us to check for the possible confusion between messages belonging to different subprotocols.

Decisions as to whether or not to execute options such as including identities in Quick Mode messages are specified in a way similar to that used for specifying the choice of subprotocols.

Finally, one minor change to the specification language had no effect on ease of use or expressiveness, but was very helpful in increasing the efficiency of searches. Originally, values that were supposed to be unique, such as nonces and keys, were identified by a name-time stamp, with name and time as argument. Thus, a key generated by server S at local time N is different from a key generated by server T at local time M unless S = T and N = M. This was a useful way of ensuring uniqueness. But it had the following undesirable result. Suppose that the Analyzer was looking for two or more terms involving unique terms identified with name-time stamps that were generated during the same round. The Analyzer would have no way of knowing this simply by looking at the terms alone, and it would often generate a new round for each such term, whether or not that was appropriate. Thus, if the Analyzer was trying to find rand(ts(Name,Time1)) and rand(ts(Name,Time2)), it would often assume that rand(ts(Name,Time1)) was generated during Round R1, and rand(ts(Name,Time2)) was generated during Round R2. Searching backwards through these two rounds doubled the effort required by the Analyzer. This could be wasteful if at the end the Analyzer discovered that the two terms could only have been generated during the same round. We managed to eliminate much of this waste by including the round number in the name-time stamp. Now, the Analyzer could tell that it was only necessary to look through one round to produce both terms.

The redesigned specification language helped greatly in the writing of understandable protocol specifications. However, it was still necessary to provide some assistance to the specification writer in determining whether or not the protocol specified was the protocol intended. This was provided by the implementation of a "sanity checker." To use the sanity checker, the user specifies a set of sequences of transitions that should occur in a legal execution of the protocol (that is, an execution in which the intruder is absent). The sanity checker then determines if these sequences can be executed. This can be used to catch a number of protocol specification errors.

We are also investigating the integration of the NRL Protocol Analyzer with Millen's Common Authentication Protocol Specification Language (CAPSL) [21]. In particular, a translator from CAPSL to the NRL specification language has been developed [3]. Although we found some discrepancies between the NRL model and the version of CAPSL that we used (discussed in [3]) that made it impractical for us to use the translator to assist us in producing the entire IKE specification, we did find it very helpful when we needed to perform a rapid specification and analysis of a subprotocol that we had left out of the original subset of IKE that we had decided to examine. We hope to continue this integration work, and, if so, we expect it to simplify the specification process even more.

4.2 Improvements to the Theorem Prover

4.2.1 Strategy for Improvement

The theorem prover in the Analyzer works in two stages. In the first stage, it is used to generate lemmas that describe under what conditions states are reachable. In the second stage it generates states, and applies the lemmas to determine which states are unreachable and should be discarded, which states are reachable only under certain conditions and should have those conditions included, and which can be kept as is.

There are two ways in which this process can be improved. One is to increase the automated assistance offered during the lemma generation process. The other is to speed up the "generate-and-test" process, which can be slow for large protocols, since tests are expensive and many more states fail the test than pass it.

The improvements to the lemma generation process came first, so we begin by describing them.

4.2.2 Improvements to the Lemma Generation Process

Very few improvements needed to be made to the language generator, since this was already almost completely automated. Most changes involved changing

the way languages were stored, in order to make it easier for the Protocol Analyzer to store more complex languages of the sort that were generated in the IKE analysis.

The State Unifier was another case altogether. In its earlier form, the State Unifier was used by looking at states generated by the Analyzer and manually constructing input based on the information contained in the states. This was a tedious and tricky process at best, and one that was also difficult to teach. When applied to a protocol suite like IKE, it became impossible.

The solution was to look closely at the techniques used to generate input for the State Unifier, and to find large subclasses which could be automated. As it turned out, there were two which could be handled this way.

The first mechanism we call the Input Evaluator. The Input Evaluator takes the state variables that are input to each rule, and works backwards to the initial state for the principal executing the rule. It keeps track of any assignments that are made to the variables during this backwards search, and gives this information to the State Unifier.

To see how this works, suppose that a rule requires as part of its input that the value of the state variable init_nonce be Y for some variable Y. Suppose furthermore that the Input Evaluator searches backwards and finds that, along all paths, the state variable init_nonce is assigned the value rand(ts(Name,Round,Time)) for some variables Name, Round, and Time. It hands this information to the State Unifier, which puts it in its database. Now, whenever the Analyzer encounters the state variable init_nonce, it will attempt to see if its value can be made to take the form rand(ts(Name,Round,Time)). If it can't it will record the state as unreachable.

The second mechanism we call the Forwarding Lemma Generator. This makes use of an idea of Paulson's [23, 25]. In his work, a forwarding lemma is used to handle the situation in which a party receives a term in a message, and then passes it along in a latter message. In such a case the intruder does not learn anything new from seeing that term, since it already saw the earlier message.

The older version of the Analyzer also dealt with this problem. Suppose that the Analyzer is trying to find out how the intruder can find a term or list of terms. It remembers the list of terms that is being searched for, and whenever it produces a state in its search, it examines the terms input. If any of the input terms is also in the list of searched for terms, that state is discarded. This works in a way similar to Paulson's forwarding lemmas, but has the disadvantage that it must be recalculated every time the situation is encountered. If we could prove lemmas stating that the state is unreachable in the first place, this would save the Analyzer a lot of unnecessary work.

The Forwarding Lemma Generator implements this strategy. It is used in two phases. In the first phase, the user uses the Analyzer to find a generic term represented by a variable X. In the second stage, the Analyzer examines the search tree and finds all states that required X as input. Whenever it finds such a state, it generates the appropriate lemma.

4.2.3 Improvements to the Generate-and-Test Strategy

When one is using a rule-based system in which the rules are subject to a large number of tests, it is helpful to have metatests that can be used to tell us which rules can be discarded immediately. In the case of the NRL Protocol Analyzer, these would be metatests that tell us which specification rules can be used to produce a given state. The Protocol Analyzer works by attempting to match output of protocol rules against state descriptions. The time necessary to match up each protocol rule against a state description and then to decide whether to accept or reject it is significant.

A state description in the NRL Protocol Analyzer consists of three things: a list of values of internal state variables, a list of event statements describing transitions that have occurred, and a list of terms known by the intruder (that have either been passed as part of messages or produced by the intruder itself). In general, a given internal state variable can only have been produced by a handful of rules. A given event statement can only have been produced by the rule it describes. A given intruder-known term, however, could possibly have been produced by a number of protocol rules which included that term as part of a message. Thus we concentrated on determining which rules could produce which intruder-known terms.

The strategy we developed, which we encoded in a procedure called genpossiblerules, is simple. First, genpossiblerules looks for intruder-known terms that the Analyzer would be likely to look for. These includes all intruder-known terms that could be accepted as input to rules, and all terms that could be generated using available intruder operations. For each such term, it tries each protocol rule to see if that rule could produce that term. If it can, it records that information in the record possiblerules(W, R), where

W is the term and R is the rule.

Later, when the Analyzer is attempting to find out how a term X appearing in as a term known by the intruder in a state description could be produced, it checks the possiblerules database. For each term W appearing in a possiblerules record such that W subsumes X, it tries *only* those rules R appearing in some possiblerules(W, R) to determine if R could have produced X. If X is subsumed by more than one such term W, say W and W', it tests only those rules R such that possiblerules(W, R) and possiblerules(W', R) appear. If X is subsumed by no term appearing in the possiblerules database (a rare event), the Analyzer tries all rules.

The genpossiblerules procedure works best when applied after the lemma generation, so it does not speed up the lemma generation process itself. However, it provides significant speedups during the path generation phase of the analysis. Moreover, it appears to provide better assistance the more complex a protocol becomes; for example, when we tried it out on a specification of the Needham-Schroeder public-key protocol, its use speeded up the analysis only by about 10 per cent; however, when we applied it IKE, the speed of the analysis was increased by a factor of about three. We expect that its use on more complex protocols would provide even greater speedup.

4.3 Improvements to the Query Structure

The user presents a query to the Analyzer by specifying a state and a search strategy. A search strategy tells the Analyzer how to behave when querying a state in the search path. The Analyzer views a state as a collection of state variables and of terms known by the intruder. It can either attempt to find all of these state variables and intruder-known terms, or it can regard some as "trivial," that is, not worth searching for because they could clearly be found. Examples of "trivial" terms would be terms known initially by the intruder, or terms represented by variables (for which the intruder could substitute any term it knows). Refusing to search for trivial terms greatly increases the efficiency of the search and does not affect the soundness of a result, since an unreachability result holds whether or not some terms were not searched for. The worst that could happen is that we might judge wrongly about the "triviality" of a term, that is, it might really be impossible for the intruder to find. In that case, the Analyzer would generate a false attack. However, false attacks are easy to recognize, and we when we find one we can always redo the search with the term mistakenly judged as "trivial" included back in the search.

The insight obtained here was that, for the purpose of efficiency, it was sometimes possible to treat terms that were clearly not trivial as if they were. This would generate a fast search, possibly with some attacks. One could then determine whether or not the attacks were valid. If they were, then we were done, and in much less time than it would have taken to perform a complete search. It the attacks were not valid, we would merely have to redo the search, but this time treating the previously "trivial" terms as non-trivial.

For example, for a protocol using digital signatures, an attack might depend on the intruder's ability to produce signed messages. Terms that were not signed messages might or might not be as useful to intruder. Thus, when analyzing such a protocol, it might make sense to ask the Analyzer to treat as trivial all terms that the intruder is required to know except those representing signed messages. Thus, in a protocol making use of concatenation, digital signatures, and hash functions, we might begin by asking the Analyzer to treat as trivial all terms encountered of the form (X,Y) (for concatenation) and hash(X) (for hash functions), thus having it query only intruder-known terms of the form pke(privkey(U),X) (for signed messages) and state variable values. This could greatly simplify the search.

As a result, we decided to give the user the option of specifying which types of terms the Analyzer treats as trivial. The user can now give the Analyze a list of intruder-known terms to be treated as trivial; the Analyzer will do so for any term subsumed by one of the terms in the list. This simple addition to the Protocol Analyzer turned out to be invaluable. It greatly reduced the time required to analyze simple protocols of the type that usually appear in the protocol analysis literature. In the case of more complex protocols suites such as IKE, it together with the use of genpossiblerules were the changes that made protocol analysis possible. Queries that had formerly resulted in unmanageable state explosions could now be processed easily. Even when discarding certain intruder-known words as trivial produced invalid attacks, we were sometimes able to use another feature in the Analyzer to produce useful information. This was a "backtracking" function that would delete all children of a node but keep all substitutions that were made to variables in a search. (If different substitutions were made along different paths, then the node would be replace by several nodes). The substitutions thus made would often narrow the scope of the search so that it became

feasible to include more types of intruder-known terms as nontrivial.

5 Specification of the IKE Protocol

In specifying the IKE protocol, we had to perform a certain amount of abstraction and simplification. Some of this was in order to deal with the then limitations of the Protocol Analyzer, and some of this was because certain features of the protocol (such as the contents of the security associations and the techniques for negotiating them) were outside the scope of our analysis. We describe our specification below.

First of all, we specified most of the subprotocols. New group mode posed some technical difficulties, which, given the other challenges posed by this protocol, we decided to avoid dealing with at this point. This was a result of the fact that the old group was used in the protocol that defined the new group. Analysis of this would introduce a possible infinite regression. The Analyzer can handle other types of infinite regressions, and it is very likely that, with a little thought, it could be made to handle this type. However, we decided that we had enough on our hands without trying to confront this now. Also, the second form of public key authentication for main and aggressive mode did not appear until relatively late in the IKE protocol design, after we had written our initial specification. Thus it was not included either. However, all of the remaining subprotocols were specified, giving a system with ten subprotocols. This we decided was more than enough to give a challenge to the Protocol Analyzer.

The commutative properties of the Diffie-Hellman algorithm also proposed a problem. As we have mentioned earlier, an Analyzer specification must specify explicitly the cryptographic identities assumed, and all such identities must be expressible as rewrite rules. However, the Diffie-Hellman algorithm relies explicitly on the commutivity of exponentiation to achieve its goals; there is really no way to abstract it away.

Originally, we experimented with an extension that introduces commutivity in a limited way, enough to include the Diffie-Hellman protocol as a special case. However, although we were able to use this to analyze a specification of the Station to Station protocol, we found it too slow to analyze IKE. We thus decided on the following compromise. We developed two sets of operations and rewrite rules, one for initiator and one for responder. In other words, computation of the Diffie-Hellman key was assumed to involve different operations for initiator and responder, even though in fact they are both the same.

For both initiator and responder, we had the operator $\exp(G, X)$, where G represents a choice of group and generator, X is the exponent, and exp is the exponentiation function. Thus $\exp(G, X)$ represents Z^X mod group(G), where Z = generator(G). For the initiator, we then used the operation $\mathrm{h}(G, W, Y)$ to represent W^Y mod group(G). where W is the value received from the responder, and Y is the initiator's exponent. For the responder, we used $\mathrm{g}(G, W, Y)$ to represent W^Y mod group(G), where W is the value received from the initiator and Y is the responder's exponent. We then use the two rewrite rules:

h(G,exp(G,A),B) → dhkey(G,A,B)
g(G,exp(G,B),A) → dhkey(G,A,B)

The reader can verify that, if initiator and responder exchange their information correctly, then the rewrite rules will guarantee that they agree on the same key. Thus, the functions we use obey *some* of the rules algebraic rules associated with Diffie-Hellman, including the ones necessary for the computation of a Diffie-Hellman key, but not all of them. Hence any attack we find will remain valid under the full set of algebraic identities, but it is possible that an unreachability proof will not.

This compromise allows us to use the Analyzer in its current state to analyze the IKE protocol, but it does have one disadvantage: it prevents the Analyzer from finding certain classes of attacks, namely those in which a key generated by two principals is confused with the same key generated by the same principals, but with their roles reversed. Such an situation could arise, for example, if two parties agreed to share a key with each believing that it was acting in the role of the initiator. We note that such attacks on IKE appear to be unlikely, as a considerable amount of care appears to have gone into assuring that different types of messages (in particular, initiator and responder messages) are always distinguishable from each other. In particular, negotiated keys are computed using a hash in which a nonce generated by the initiator precedes the nonce generated by the responder, so two responders would compute different keys. Since the negotiated keys are used to authenticate the last messages in a Phase One Exchange, this should prevent against this kind of attack in Phase One. Also, the keyed hashes used for authentication in the Phase Two protocols are computed differently for initiator and responder, and so such a confusion attack is probably not very likely for Phase Two either. However, our use of a noncom-

mutative representation of Diffie-Hellman does represent a limitation in our analysis, and should be noted.

Also, about halfway through the analysis we divided the specification into two parts which we analyzed independently. The first part contained all the Phase One protocols, and the second part contained all the Phase Two protocols. This was done because we found that the analysis was taking too long otherwise. However, it was also done before we introduced the genpossiblerules operator. We expect that if we analyzed the original protocol using that operator, we would have much better results, and would be able to analyze it in its entirety. However, given the lack of negative interaction that we found between the much more closely related protocols within Phase One and Phase Two, we do not expect that such an analysis would produce any new information.

The security association negotiation was also greatly abstracted and simplified, as we did not want to get involved in the details of the contents of the security associations and their negotiation. In particular, we assume that, although the initiator can propose a number of security associations, the responder can only pick one. We thus were faced with the problem of specifying indeterminate terms, and indeterminate choices made from indeterminate terms. We solved this problem by using free variables to represent nondeterministic choice of terms. For example, a list of security associations was represented by the term plist($PROPOSALLIST$) where $PROPOSALLIST$ was a free variable. Thus the fact that two lists were the same could be expressed by unifying the two free variables used to define them. Similarly, a member of a security association could be expressed as choice(N,plist($PROPOSALLIST$)) (or more compactly, choice(N,$PROPOSALLIST$)) where N was another free variable representing the number of the security association in the list. Finally, the choice of a particular element of a security association in a list could be expressed as choice(N,$TYPE$,$PROPOSALLIST$). The choice of $TYPE$ was usually deterministic, and would be represented by a constant rather than a free variable. This allowed us to represent a list of proposals without having to specify its contents completely. We also assumed that a responder always accepts one of the security associations proposed.

We also did not try to include much information about what exactly a peer and an identity was, other than that a peer was a party taking part in the protocol, and an identity identified some party on whose behalf the peer was negotiating, possibly but not necessarily the peer itself. That is because the meaning of these terms was purposely left vague in the ISAKMP specification itself, since ISAKMP is intended to have a wide range of applications. For example, the protocol could be used for communication within a virtual private network, to protect the communications between a virtual private network and the rest of the world, to protect traveling employees who want to log in to their home system while on the road, or to help small groups of users set up "webs of trust" [14]. Each of these applications may require different definitions of what a peer is and what an identity is. As it turned out, this underspecification of identities in ISAKMP led to an ambiguity in the IKE documentation which our analysis caught.

Finally, we did not include a mechanism that IKE uses to protect against interleaving and reflection attacks when cipher block chaining is used. This is to use the last cipher block of an encrypted message as the initialization vector of the next message in the exchange. We omitted this because including it would require a specification of cipher block chaining, and we did not want to include this level of detail. However, whenever we did find an anomaly or undesirable behavior, we checked to see if the use of this mechanism would have prevented it.

We end with a note about key compromise. We note that key exchange protocols are designed to be secure against attacks involving compromise of old keys. Thus we needed to introduce the possibility of key compromise into the specification. Hence, whenever an event in which a party accepted a key occurred, we included an event occurring immediately afterword in which that key either was or was not compromised, using nondeterministic choice. In the Phase Two Protocol, both master and session keys could be compromised.

6 The Analysis of the IKE Protocol

6.1 Overview of IKE Analysis

One of the goals of our IKE analysis was to determine if their were any harmful interactions between the IKE subprotocols. We found none. IKE appears to be very well-designed from this point of view. However, we did find a few problems that arose from omissions and inconsistencies in the IKE specification. We describe these and our other findings below.

There are a number of types of questions one can ask about a key agreement protocol, but there are three in particular that are common to all such proto-

cols. The first is secrecy: can an intruder learn a secret key that is shared between two honest principals without a compromise event having occurred? The second is authentication, namely the requirement that if a key is accepted for some purpose, it should have been generated for that purpose: if the principal B who receives the final message in the protocol accepts key K (or in this case, the entire security association SA) as good for communication with A, then A should have also have accepted SA as good for communication with B. The third we may call *relevance*; if a key is accepted by an honest initiator (resp. responder) A who receives the final message in the protocol as good for communication with an honest responder (resp. initiator) B using security association SA, then the only party who could have accepted it previously is B acting as responder (resp. initiator) as good for communication with initiator (resp. responder) A using SA.

We use the following lemma to make the work of verifying relevance easier once we have verified authentication.

Lemma: Assume that a protocol satisfies authentication and secrecy. Then, if we can show that a key is never accepted by two parties acting in the same role (who may or may not be the same party accepting the key twice, e.g. a replay attack), then the protocol also satisfies relevance.

Proof. We may assume without loss of generality that the responder is the last party to accept the key in the protocol specification. Suppose that relevance is not satisfied. Let B be a responder who has accepted a key K as good for communication with a responder A, using security association SA. By authentication, A as initiator must have also accepted K as good for B and SA. For relevance to fail to hold, another event in which some party accepted the key must have occurred. Since that party must be either initiator or responder, the key must have been accepted by two initiators or two responders.

We also verified that the version of the Phase Two protocol that used Diffie-Hellman key exchange satisfied perfect forward secrecy, since this was one of the stated goals of that protocol. Perfect forward secrecy is the property that, if a master key is compromised, the intruder does not learn any session keys established before the compromise, assuming that the session keys themselves are not separately compromised.

All of our analyses were performed in the same manner. For each specification, we proved a number of initial lemmas describing conditions under which different terms and state variables could be produced. This followed a standard procedure that we use in the analysis of all protocols. First, we used the evalnewinput command to produce conditions on the reachability of internal state variables. Then we used the Forwarding Lemma Generator command to produce forwarding lemmas. Next we used the Language Generator to produce languages, using the standard seed terms. Finally, we used the genpossrules command to determine what protocol rules could produce what terms.

Once this was done, we were ready to begin the analysis itself. We formulated each security goal in terms of conditions on sequences of events, and then took the negation of that goal to present to the Analyzer. Thus, for example, if we wanted to show that in order for an initiator A to accept a key as good to communicate with a responder B, then B must have accepted a key as good for communication with A, we would ask the Analyzer if it was possible for A to have accepted the key without B having accepted it.

Once we had formulated the goal and presented it to the Analyzer, the rest of the search was almost completely automatic. The Analyzer would search backwards from the goal in a breadth-first fashion, terminating any path in which it encountered an initial state or a state previously shown to be unreachable by one of the lemmas we had proved earlier. Usually, the only further input that was required was to tell the Analyzer which types of intruder-known terms in a subgoal should be discarded as trivial. In general, we always told the Analyzer to discard certain types of nonces, (e.g. cookies) as trivial, since they could be easily learned by the intruder. In several cases, we also told the Analyzer to discard everything but signed and encrypted messages as trivial when we felt that the search space was getting too large otherwise. Occasionally, we would find that discarding so many types of intruder-known terms produced false attacks, in which case we would use the backtracking function and try again. Sometimes we would find that the same unreachable state kept on cropping up in our analysis; we would reduce the search space by proving that state unreachable and adding the result to our database of lemmas using the State Unifier. In general, though, we avoided using the State Unifier, since searching for the appropriate input was time-consuming.

6.2 Analysis of Phase One

6.2.1 Questions Put to the Analyzer

In the case of the Phase One exchange, we asked whether the protocol satisfied secrecy, authentication

and relevance. We also asked about a property related to authentication: under what conditions would the sender (as opposed to the receiver) of the final message in the protocol accept a security association SA. For relevance, we used our authentication results and the Lemma to show that we only needed to show that two initiator accept events or two responder accept events could not occur.

6.2.2 Ambiguous Specification of Identities

Before our analysis was fully underway, we found the following anomaly. The phase one protocol is defined as a communication between two peers that exchange identities. Before the identities are exchanged, the peers can only be identified by their IP addresses. Thus the question remains: what should the authentication keys be bound to, the identity or the IP address? In most modes of IKE, it appeared possible to do either, but in the case of main mode with shared key, ID's are exchanged only after the keys are used. Thus in that case, keys must be bound to the IP address. Initially, in order to simplify our analysis, we assumed that keys were bound to the IP address in all cases.

As we proceeded in our analysis, we found that the Analyzer generated a number of attacks, all of which could be thought of as variants of the following. Alice lives on secure host X, and Bob lives on secure host Y. Eve from compromised host Z manages somehow to convince Alice that Bob lives on Z. Alice initiates a key exchange with Bob on Z. Since Eve has compromised Z, she is easily able to impersonate Bob to Alice. Similarly, Eve initiates a key exchange with Bob as Alice from Z. Now Eve is able to initiate a man-in-the-middle attack in which she receives traffic from Alice, decrypts it, re-encrypts it with the key Eve shares with Bob, and so forth.

When we pointed this out to the IKE and Oakley designers, we found the problem rested on a misunderstanding of the intent of the IKE specification. Keys are intended to always be bound to identities. If keys are to bound to IP addresses, then the identities must be based on the IP address. However, this was not made clear in the IKE specification. As a result of our comments, the designers of IKE decided to put a clarification into the IKE specification.

We continued our analysis using the assumption that keys are bound to IP addresses, and that identities are IP-based. This did not capture all possibilities, but it did allow us to go on and attempt to analyze the security of the rest of the system without worrying about this particular issue.

6.2.3 Penultimate Authentication

From this point on our analysis proceeded fairly smoothly. In the case of our first question, about the conditions under which the receiver of a final message would accept a security association, the Analyzer found that indeed there was no case in which the receiver A of the final message accepted a security association SA as good for communication with B without B having itself accepted SA. Likewise, we were able to show relevance, that the only two accept events involving the key generated for SA were A and B accepting SA. The case of what happens when the sender B of the final message accepts a security association as good for communication with A was a different matter, however. What we wanted to show was that, if B accepted an SA as coming from A, then A had also accepted that SA (minus the keying material contributed by B if that had not been sent yet). We will refer to this property as "penultimate authentication," since it describes desirable behavior at the penultimate stage of the protocol. The Analyzer found that penultimate authentication was not always guaranteed; all of the attacks against it that it found involved the intruder confusing the two communicating parties about the nature of the identities being exchanged.

That penultimate authentication was not guaranteed was not surprising; as a matter of fact it had already been shown that the Station to Station protocol, which was one of the protocols that influenced IKE, did not satisfy penultimate authentication either, and for similar reasons [11]. Nor is the lack of penultimate authentication necessarily a serious concern: if B accepts an SA as coming from A which A did not in fact initiate, then it will send a message to A which A will reject. Since the intruder does not in fact learn the corresponding key, that means that the worst that can happen is that B accepts an SA for communication with A that will never be accepted by A. Since the same thing can happen if the intruder simply prevents B's final message from reaching A, the threat posed by any lack of penultimate authentication depends on whether or not it is believed to be easier to fake a penultimate message than it is to prevent a final message from reaching its destination.

On the other hand, penultimate authentication is of some interest because of its close relation to the fail-stop property of Gong and Syverson [5] which can be roughly defined as a property that requires that, if a principal receives an incorrect message, then it should be able to to detect it and refuse to process it. Fail-stop protocols are desirable because of their

composability properties, and because of the greater ease with which it is possible to reason about them. A protocol that lacks penultimate authentication is probably as far as it can be from being fail-stop while still satisfying its security goals.

It was also the case that whether or not penultimate authentication failed to hold and the degree to which it failed to hold depended upon the type of authentication and mode used. For shared key in both main and aggressive mode, penultimate authentication always holds. For digital signatures in both main and aggressive modes and public keys in aggressive mode, the property never holds. For public keys, it fails to hold in main mode when the sender of the penultimate message also initiates a conversation with a dishonest principal.

We also conjectured that the revised public key protocol offered in IKE would satisfy penultimate authentication, since it binds identities to the keying material by encrypting them both with the same key. Although we did not have time to include this protocol in our IKE specification, we used the Analyzer to perform separate quick and dirty evaluations of simplified versions of both the aggressive and main modes of the protocol that omitted such features as the negotiation of security associations and the details of key derivation[2]. We also used our commutative definition of Diffie-Hellman key exchange, instead of the non-commutative version that we used for the larger specification. The Analyzer proved both that both specifications satisfied penultimate authentication. Thus we can conclude that, if penultimate authentication is desired, it should be possible to achieve it within the IKE framework by choosing the appropriate Phase One key agreement protocol.

Lack of space prevents us from presenting all the penultimate authentication failures that we found. However, we will give one example. using digital signatures in aggressive mode, to give an idea of the flavor.

The protocol itself proceeds is follows

1. $A \rightarrow B \ : \ HDR_1, SA_A, KE_A, N_A, ID_A$

   where HDR_1 is the message header, SA_A is the security association proposed by A, KE_A is A's Diffie-Hellman key material, N_A is a nonce, and ID_A is A's identity.

2. $B \rightarrow A \ : \ HDR_2, SA_B, KE_B, N_B, ID_B,$
   $K_B{}^{-1}[prf(K_{AB}, (KE_B, KE_A, CKY_B$,
   $CKY_A, ID_B)]$

   where prf is a pseudo-random function, K_{AB} is the Diffie-Hellman key generated from KE_A and KE_B, and CKY_A and CKY_B are a pair randomly generated cookies generated by A and B respectively and included in the headers.

3. $A \rightarrow B \ : \ HDR_3,$
   $KA^{-1}[prf(K_{AB}, (KE_A, KE_B, CKY_A,$
   $CKY_B, ID_A)]$

The "attack" proceeds as follows:

1 $A \rightarrow B \ : \ HDR_1, SA, KE_A, N_A, ID_A$

The intruder intercepts this message and substitutes ID_K for ID_A, forwarding the result to B.

1' $I_K \rightarrow B \ : \ HDR_1, SA, KE_A, N_A, ID_K$

where I_K stands for intruder I impersonating K.

2' $B \rightarrow K \ : \ HDR_2, SA, KE_B, N_B, ID_B,$
$K_B{}^{-1}[prf(K_{AB}, (KE_B, KE_A, CKY_B$,
$CKY_A, ID_B)]$

This is also intercepted by the intruder and forwarded to A.

2 $I_B \rightarrow A \ : \ HDR_2, SA, KE_B, N_B,$
$ID_B, K_B{}^{-1}[prf(K_{AB}, (KE_B, KE_A, CKY_B$,
$CKY_A, ID_B)]$

A takes this as a message from B in response to its initial message. But B thinks that it is K who initiated a response with it, and when it receives a signed message from A it will reject it.

6.3 Analysis of Phase Two

Finally, we performed an analysis of IKE Quick Mode, which comprises the protocols used for a Phase Two Exchange. Here we asked only three questions: did the protocol satisfy secrecy, did it satisfy authentication, and did it satisfy perfect forward secrecy. We did not ask about relevance, because, as it turned out, we found an attack authentication attack on our specification that was also a relevance attack.

Both master and session key compromise were specified in our specification of the Phase Two protocol; this meant that we had to formulate our secrecy question slightly differently than in Phase One: could the intruder discover a negotiated key given that the key had not been compromised and that the key negotiated in Phase One to protect that key exchange had not been compromised either? We formulated the perfect forward secrecy question as follows: could the intruder find a key K that had been accepted by an

[2]These were the two specifications produced with the assistance of the CAPSL-to-NRL translator.

honest responder (the receiver of the final message in the protocol) for use with an honest initiator using master key M, assuming that no compromise event involving K occurred, and no compromise event involving M occurred before the acceptance event? The answer to both these questions was in the negative: our specification satisfied both secrecy and perfect forward secrecy.

But it was in the analysis of the authentication question that the Analyzer found our most interesting result: a potential attack in which B thinks it is sharing an SA with A, when actually it is sharing it with itself. This attack turned out to be foiled by some implicit implementation assumptions in IKE, but these were not stated clearly in the specification. Once we found this attack, we did not bother asking questions about relevance, because the attack was a violation of relevance as well.

To understand the attack, we must recall that the inclusion of identification information in a Quick Mode exchange is optional when the identities correspond to IP addresses. In that case, the receiver B of the message can use the IP address of the sender A of the message as an index to the encryption key. But this gives B no way of telling messages from A from messages from B. All the intruder needs to do is substitute A's IP address for B's.

We describe IKE Quick Mode (without the identities) and the attack below. B begins by creating a unique message ID M_B using a random or pseudo-random number generator. M_B is used to identify all further messages exchanged in an single execution of the protocol, and is included unencrypted in the header of each message. The key K_{AB} is the encryption key generated during Phase One.

1. $B \to A:\ HDR_1$,
   $EK_{AB}[prf(AK_{AB}, (M_B, SA_B, N_B, KE_B)),$
   $SA_B, N_B, KE_B]$

   where HDR_1 is the header of the message, (which contains M_B, among other information), EK_{AB} is the encryption key shared between A and B as the result of the Phase One exchange, AK_{AB} is the authentication key shared between A and B as a result of the Phase One exchange, SA_B is the security association proposed by B, N_B is a nonce generated by B, and KE_B is the optional Diffie-Hellman key material sent by B.

2. $A \to B:\ HDR_2,$
   $EK_{AB}[prf(AK_{AB}, (M_B, N_B, SA_A, N_A, KE_A)),$
   $SA_A, N_A, KE_A]$

   where SA_A is the security association proposed by A, N_A is the nonce generated by A, and KE_A is the optional Diffie-Hellman key material generated by A. Note that the second message is syntactically identical to the first, except that the prf is computed over two nonces instead of one.

   When B receives the above message, it can now compute the shared key, which is generated using N_B, N_A, KE_A and KE_B, if included, and keying and other material generated during the Phase One Exchange.

3. $B \to A:\ HDR_3, prf(AK_{AB}, (M_B, N_B, N_A))$

   At this point A also computes the shared key.

The attack proceeds as follows:

1 $B \to A:\ HDR_1,$
$EK_{AB}[prf(AK_{AB}, (M_B, SA_B, N_B, KE_B)),$
$SA_B, N_B, KE_B]$

This message is intercepted by the intruder.

1' $I_A \to B:\ HDR_1,$
$EK_{AB}[prf(AK_{AB}, (M_B, SA_B, N_B, KE_B)),$
$SA_B, N_B, KE_B]$

where I_A denotes the intruder I impersonating A.

2' $B \to A:\ HDR_2,$
$EK_{AB}[prf(AK_{AB}, (M_B, N_B, SA'_B, N'_B, KE'_B)),$
$SA'_B, N'_B, KE'_B]$

This message is also intercepted by the intruder.

2 $I_A \to B:\ HDR_2,$
$EK_{AB}[prf(AK_{AB}, (M_B, N_B, SA'_B, N'_B, KE'_B)),$
$SA'_B, N'_B, KE'_B]$

At this point B generates a key K and accepts it as a good key for communicating with A.

3 $B \to A:\ HDR_3,$
$EK_{AB}[prf(AK_{AB}, (M_B, N_B, N'_B))]$

This message is intercepted by the intruder.

3' $I_A \to B:\ HDR_3,$
$EK_{AB}[prf(AK_{AB}, (M_B, N_B, N'_B))]$

B generates key K again and accepts it as a good key for communicating with A.

The end result of this is that B winds up thinking that it shares two keys with A, when actually it does not share any keys at all. As a matter of fact, A does not send or receive a single message! Thus B has been denied a service (a shared key with A) without even being aware that the service has been denied.

Once we found this attack, we took a closer look at the IKE specification to see if there were any implementation details that we had overlooked. First of all, we looked at the protection IKE offered against reflection attacks in general. We found that IKE's technique of using the last ciphertext block of each message as the initialization vector of the next message in an exchange does not help us here. This technique is intended to provide protection against attacks in which a reflected message is inserted into an ongoing exchange, but does not provide any protection against an the reflection of an entire message sequence, which is what happened in the attack described above.

We also considered the possibility that the protocol encoding scheme used might require authentication of the IP addresses, or that separate inbound and outbound keys might be used. Either or these would foil this attack. However, we could find no place in the IKE specification where either of these were recommended for Phase Two exchanges, and our discussions with others lead us to believe that no such requirement was intended.

We finally found the way IKE protects against our attack after some discussion with several people connected with IKE, and another look at the ISAKMP documentation. ISAKMP first requires that the message ID be random, so that the probability of two message ID's being the same is remote. Secondly, the ISAKMP header contains no other indication of whether a message is an initial message or a response to another message. Thus, when a principal receives a Quick Mode message, there are only two ways in which it can determine whether or not it is an initial message. One is to decrypt the message and examine its contents. Note that, since the first two messages of a Quick Mode exchange are syntactically identical, this examination must also include verifying the hashes. The other is to check if there are any other ongoing exchanges with the same message ID. If there are, the messages is assumed to be a response. If there are none, it is assumed to be an initial message.

Clearly, the second way of determining the provenance of a message would prevent the attack we discovered. When B received the reflected message, it would check for the message ID and conclude that it was a response to its original message instead of another initial message. When it decrypted the message, it would realize that it was the wrong message and reject it. Moreover, if cipher block chaining was used, it would use the last cipher block of the message as the initialization vector, thus producing gibberish from the first cipher block when it tried to decrypt it.

However, there were two problems. First of all, although the ISAKMP documentation clearly states that message IDs should be randomly generated, there is no such language in IKE. This omission had already led to several implementations in which message IDs were generated by simple counters. Secondly, although the IKE documentation does say that message IDs are to be used to keep track of Quick Mode exchanges, it does not say explicitly how they must be used. Thus, it could be possible to implement checking message provenance in another way, especially if the ISAKMP header is ever modified to include more information about the place of a message in the exchange. As a result we made two recommendations. Once was to include in IKE the requirement that message IDs be randomly generated. The other was to include an explicit description of the way provenance of quick mode messages must be checked. We have been informed that these issues will be addressed in future iterations of the IKE document.

7 Conclusion and Lessons Learned

It is interesting to note that almost all of the problems we found in IKE came from a single source: the omission of identities in various parts of the protocol. These omissions were no accident; IKE is deliberately designed so that the user has the option of not giving out identification information until it is absolutely necessary. This feature, which is designed to protect the privacy of negotiations while a protocol is completing, is somewhat at odds with the requirements of authentication, and it is not surprising that it is tricky to design and specify a protocol that must satisfy such conflicting goals. Thus our formal analysis of IKE was mainly useful in establishing that the conflicting requirements were handled correctly, and in pointing out ambiguities in the specification that could lead to the requirements being violated. Problems involving conflicting requirements have often been a fruitful area of application for formal methods; we should not be surprised that the same turned out to be the case here.

The analysis gave us valuable information, not only about IKE, but about possible new directions for the NRL Protocol Analyzer. Some of these were relatively minor. For example, we found that we were unable to specify the correct implementation of the use of Message IDs in Quick Mode, because of limitations on the way conditions on states can be expressed in Analyzer specifications. In order to specify the determination that a Quick Mode message is initial, it is necessary to be able to specify that there should be no local

state variable in a current exchange with the same message ID. But, although it is possible to put conditions on state variables belonging to the specified round in an Analyzer specification, it is not possible to put conditions on all possible state variables belonging to a principal. We are investigating ways in which the specification language could be extended to include this in a sound and efficient way.

Other possible directions are more far-ranging. One of these is extension to denial of service. In our analysis of IKE, we did not attempt to find denial of service attacks, although denial of service was a concern of the IKE designers. Indeed, in the current state of the Analyzer it is impossible to model denial of service directly. This is because, in the Analyzer model, denial of service is trivially possible: the intruder could simple block all messages and prevent a protocol from completing. However, we have recently developed a model of denial of service [19] that incorporates some of the ideas used in the design of the Photuris protocol [7] and further developed by Kent et al. [9]. This model involves trading off different protocol properties against each other, and protection against intruders of various levels of strength at different points in the protocol, so it is beyond the scope of existing protocol analysis tools at the moment. However, we believe that it should be straightforward to modify them so that they can evaluate protocols according to the model, and in [19] we outline a strategy for doing so.

Finally, there is the central problem we attempted to deal with in this work: the analysis of protocols that include multiple subprotocols. We have shown that this is possible at least on a relatively small scale; but this is only a beginning. Previously, concern about related protocol attacks were generally relevant to subprotocols of a protocol defined in a given standard. However, with the increased use of open standards (such as ISAKMP) and common cryptographic APIs, the potential size of sets of possibly related protocols is unlimited. Thus, in order to analyze related protocols it is necessary to have some way of proving results about their possible interaction whose difficulty grows at worst relatively slowly with the size of the set.

Unfortunately, general results about the composability of cryptographic protocols are scarce, and what does exist, such as the work of Gong and Syverson [5] and of Heintze and Tygar [6], puts very strong conditions on the protocol, such as requiring that every message be authenticated for freshness and origin, and not allowing the possibility of key compromise. Moreover, there are some strong negative results, such as Kelsey and Schneier's proof [8] that for any given protocol it is possible to develop another protocol with which it interacts insecurely.

However, it is possible that some of the existing results could be extended to give us something more useful. For example, it appears that we should be able to use something like the *message independence* property of Heintze and Tygar [6]. Briefly, two protocols are message independent if no message sent during a secure execution of one protocol may appear during an execution of the other. In order for their results to hold Heintze and Tygar must introduce further restrictions on protocols that are probably not satisfied by many of the protocols we wish to examine, so we cannot use them directly. However, our use of the genpossiblerules function provides something very much like a means for testing for message independence, and it was seen to be very effective in reducing the amount of work required for protocol analysis. We plan to see how much further we wish to push this in order to provide more effective means for testing for noninteraction between protocols.

In summary, we found our analysis of IKE valuable for a number of reasons. First, it showed us that, after some modifications to the Analyzer, analysis of a collection of related protocols was possible, even when potential interactions were considered. Secondly, it motivated us to make changes to the Analyzer that not only made analysis of large protocols possible, but made the Analyzer faster and easier to use for small protocols. Thirdly, it has pointed out some promising future directions for research we plan to pursue. And last but not least, it has provided some useful feedback on and insight into IKE itself, and, we hope, a greater degree of confidence in IKE's correctness.

8 Acknowledgments

We would like to thank Hilarie Orman, Dan Harkins, and Kai Martius for helpful discussions of the IKE protocol. We would also like to thank the anonymous referees, whose comments greatly improved the presentation of this paper. This work was supported by DARPA. Portions of this paper appeared in [18].

References

[1] SET Secure Electronic Transaction specification, version 1.0. available at http://www.setco.org/, May 31 1997.

[2] J. Benaloh, B. Lampson, D. Simon, T. Spies, and B. Yee. The private communication technology protocol. draft-benaloh-pct-00.txt, October 1995.

[3] S. H. Brackin, C. A. Meadows, and J. K. Millen. A CAPSL interface for the NRL Protocol Analyzer. In *Proceedings of ASSET99*. IEEE Computer Society Press, March 1999.

[4] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 1992.

[5] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In R. K. Iyer, M. Morganti, Fuchs W. K, and V. Gligor, editors, *Dependable Computing for Critical Applications 5*, pages 79–100. IEEE Computer Society, 1998.

[6] N. Heintze and J. D. Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.

[7] P. Karn and W. Simpson. The Photuris session key management protocol. Internet draft: draft-simpson-photuris-17.txt, November 1997.

[8] J. Kelsey and B. Schneier. Chosen interactions and the chosen protocol attack. In *Security Protocols, 5th International Workshop April 1997 Proceedings*, pages 91–104. Springer-Verlag, 1998.

[9] S. T. Kent, D. Ellis, P. Helinek, K. Sirois, and N. Yuan. Internet routing infrastructure security countermeasures. BBN Report 8173, BBN, January 1996.

[10] H. Krawczyk. SKEME: A versatile secure key exchange mechanism for Internet. In *ISOC Secure Networks and Distributed Systems Symposium*, San Diego, 1996.

[11] G. Lowe. Some new attacks upon security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 162–169. IEEE Computer Society, June 1996.

[12] Z. Manna, A. Anuchitanukul, N. Bjorner, A. Browne, E. Chang, M. Colon, L. de Alfaro, H. Devarajan, H. Sipma, and T. Uribe. STeP : the Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Department of Computer Science, Stanford University, June 1994. Available at http://www-step.stanford.edu/.

[13] K. Martius. IKE protocol analysis. http://www.imib.med.tu-dresden.de/ imib/ personal/ Kai.html, Oct. 6 1998.

[14] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet security association and key management protocol (ISAKMP), version 10. draft-ietf-ipsec-isakmp-10.txt, July 3 1998.

[15] C. Meadows. A system for the specification and verification of key management protocols. In *Proceedings of the 1991 IEEE Symposium in Research in Security and Privacy*. IEEE Computer Society Press, May 1991.

[16] C. Meadows. Applying formal methods to the analysis of a key management protocol. *The Journal of Computer Security*, 1(1):1–35, 1992.

[17] C. Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[18] C. Meadows. Using the NRL Protocol Analyzer to examine protocol suites. LICS Workshop on Formal Methods and Security Protocols, 1998.

[19] C. Meadows. A formal framework and evaluation method for network denial of service. submitted for publication, February 1999.

[20] J. Millen. Comments on ISAKMP. MITRE white paper, May 1996.

[21] J. K. Millen. CAPSL: Common Authentication Protocol Specification Language. Technical Report MP 97B48, The MITRE Corporation, 1997. See http://www.csl.sri.com/ millen/capsl.

[22] H. Orman. The Oakley key determination protocol, version 2. draft-ietf-ipsec-oakley-02.txt, 1996.

[23] L. C. Paulson. Proving properties of security protocols by induction. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, 1997.

[24] L. C. Paulson. The inductive analysis of the Internet protocol TLS. Report 440, Cambridge University Computer Science Laboratory, 1998.

[25] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[26] S. Stubblebine and C. Meadows. On known and chosen cipher pairs. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*. available at http://dimacs.rutgers.edu/Workshops/Security/program2/program.html, September 1997.